# 第3章 Shell编程

Shell是UNIX/Linux操作系统中用户与系统交互的接口。它是一个命令行解释器，为用户提供了一个向Linux内核发送请求以便运行程序的界面系统级程序，用户可以使用Shell来启动、挂起、停止，甚至是编写一些程序。Shell有自己的编程语言用于对命令的编辑，易编写，易调试，灵活性较强。

Shell除了作为命令行解释器以外，还是一种高级程序设计语言，利用Shell编程可以把命令进行有机结合，形成功能强大、代码简洁的新命令。熟练掌握Shell编程可以极大地提高用户管理使用UNIX/Linux操作系统的效率。

本章从Shell的基本概述开始，介绍了Shell脚本程序设计中的语法结构、变量定义、特殊字符、控制语句等内容，并且给出了简单的实例。

本章主要学习以下内容。

- 了解Shell的基本概念、分类和功能等。
- 熟练掌握Shell脚本的建立与执行方法。
- 掌握Shell变量及特殊字符。
- 熟练掌握常用的Shell程序设计逻辑结构语句。

## 3.1 Shell概述

Linux操作系统的Shell作为操作系统的外壳，为用户提供使用操作系统的接口。它是命令语言、命令解释程序及程序设计语言的统称。

作为程序设计语言来说，Shell是一种脚本语言，而脚本语言的优点在于简单易学。相比C、Java等高级语言更能给使用者带来很大的方便。

### 3.1.1 Shell的分类

Shell作为UNIX/Linux操作系统的标准组成部分，正如UNIX版本众多一样，Shell也产生了多个版本。目前，比较常见的几种Shell如下所述。

(1) Bourne Shell：Bourne Shell是美国AT&T公司的Bell实验室的史蒂夫·伯恩(Stephen Bourne)为AT&T的UNIX开发的，于1979年年末在UNIX的第7版中推出，用作者的名字命名，简称为sh。Bourne Shell当时主要用于系统管理任务的自动化。此后，Bourne Shell更是凭借着其简单、高效的功能广受欢迎，并且很快成为当时流行的Shell。尽管Bourne Shell在Shell编程方面相当优秀，但是仍然缺少一些交互的功能，如命令作业

控制、历史和别名等。

(2) C Shell：C Shell 是比尔·乔伊(Bill Joy)在加州大学伯克利分校读书期间为 BSD UNIX 开发的，简称为 csh。其语法类似于 C 语言。此外，C Shell 提供了增强交互使用的功能，如作业控制、命令行历史和别名等。

(3) Korn Shell：Korn Shell 是 Bell 实验室的戴维·科恩(David Korn)在 20 世纪 80 年代早期开发的，简称为 ksh。它完全向上兼容 Bourne Shell 并且包含了 C Shell 的很多特性，功能更强大。

(4) Bourne-Again Shell：Bourne-Again Shell 是由 Bourne Shell 发展而来，在 1987 年由布莱恩·福克斯(Brian Fox)为了 GNU 计划编写，简称 bash。它与 sh 稍有不同，包含了 csh 和 ksh 的特色，但是绝大多数的脚本都可以不加修改地在 Bourne-Again Shell 上运行。Bourne-Again Shell 是绝大多数 Linux 发行版的默认 Shell，也是本文介绍的 Shell。

### 3.1.2 Shell 的功能

Shell 主要有两个功能，一个是命令解释器；另一个是作为一种高级程序设计语言可以编写出代码简洁、功能强大的程序。

Shell 作为命令解释器的具体功能：它接收用户输入的命令，进行分析，创建子进程，由子进程实现命令所规定的功能，等子进程终止后，发出提示符。它的作用类似于 Windows 操作系统中的命令行，但是 Shell 的功能远比命令行强大得多。

Shell 作为一种高级程序设计语言，它几乎有高级语言所需要的所有元素，包括变量、关键字、各种控制语句等，而且还拥有自己的语法结构。Shell 有自己的编程语言，用于对命令的编辑，它允许用户编写由 Shell 命令组成的程序。

Shell 另一个功能是提供个人化的使用者环境，这通常在 Shell 的初始化文件中完成(.profile、.login、.cshrc、.tcshrc 等)。这些文件包括了设定终端机键盘和定义窗口的特征；设定变量，定义搜寻路径、权限、提示符号和终端机类型；以及设定特殊应用程序所需要的变量，例如，窗口、文字处理程序及程序语言的链接库。

### 3.1.3 Shell 脚本的建立与执行

在 UNIX 或者 Linux 操作系统中，Shell 既是用户交互的界面，也是控制系统的脚本语言。用户可以通过使用 Shell 使大量的任务自动化，以此来提高系统管理的效率。

Shell 脚本(Shell Script)是指使用用户环境 Shell 提供的语句所编写的命令文件。Shell 脚本可以包含任意从键盘输入的 Linux 命令。

**1. Shell 脚本的建立**

建立 Shell 脚本的方式同建立普通文本文件的方式相同，可以利用 Linux 操作系统下的文本编辑器进行编辑工作。VI 是 Linux 操作系统下常见的文本编辑器，在终端输入命令：

```
[root@localhost ~]#vi mytest
```

进入 VI 编辑器，输入如图 3-1 所示两行程序语句，保存 mytest 文件，就完成了一个 Shell 脚

本文件的建立。


图 3-1 VI 编辑窗口

**2. Shell 脚本的执行**

1) 脚本名作为 Shell 参数的执行方法

基本语法格式如下：

```
sh script-name
```

或者

```
bash script-name
```

这种方法是当脚本文件本身没有可执行权限(文件权限执行位为-)时常使用的方法，或者脚本文件开头没有指定解释器时需要使用的方法。

例如，执行一个已经建立的 Shell 脚本 mytest，执行方式如下：

```
[root@localhost ~]#sh mytest
first shell program
2018年 01月 24日 星期三 23:11:04 CST
```

2) 修改为可执行权限的执行方法

脚本文件在建立时，其访问权限和普通文本文件一样，没有可执行权限。先用 chmod 语句将脚本文件的可执行权限加上(文件权限执行位为 x)，然后在终端直接输入脚本名称的绝对路径或者相对路径就可以。

例如，对已经建立的 Shell 脚本 mytest 加上可执行权限，然后直接执行：

```
[root@localhost ~]#ls -l mytest
-rw-r--r--. 1 root root 31 1月  24 23:10 mytest
[root@localhost ~]#chmod a+x mytest
[root@localhost ~]#ls -l mytest
-rwxr-xr-x. 1 root root 31 1月  24 23:10 mytest
[root@localhost ~]#./mytest
first shell program
2018年 01月 24日 星期三 23:11:04 CST
```

3) source 或者“.”命令

基本语法格式如下：

```
source script-name
```

或者

```
. script-name
```

第 1)、2)种执行方法都是在当前 Shell 中新建一个子 Shell，在子 Shell 中执行脚本语句；而 source 或者“.”命令(注意：“.”后面要加空格)的功能是直接在当前 Shell 中读入脚本并执行脚本语句，而不是产生一个子 Shell 来执行文件中的命令。

同样执行一个已经建立的 Shell 脚本 mytest，执行方式如下：

```
[root@localhost ~]# source mytest
first shell program
2018 年 01 月 24 日 星期三 23:11:04 CST
```

## 3.2 Shell 中的变量

在任何程序设计语言中，变量都是一个不可缺少的元素。从本质上讲，变量就是在程序中保存用户数据的一块内存空间，而变量名就是这块内存空间的地址。Shell 变量的名字可以由数字、字母和下划线组成，并且只能以字母或者下划线开头。

Shell 变量有两种类型，即 Shell 环境变量(Shell Environment Variable)和用户自定义变量(User Define Variable)。

### 3.2.1 Shell 的环境变量

Shell 的环境变量是所有的 Shell 程序都可以使用的变量。Shell 程序在运行时，都会接收一组变量，这组变量就是环境变量。环境变量会影响到所有脚本的执行结果。表 3-1 列出了常用的 Shell 环境变量。

**表 3-1 常用的 Shell 环境变量**

| 环境变量 | 说明 |
|---|---|
| PATH | 指定命令的搜索路径，以冒号为分隔符 |
| HOME | 指定用户的主工作目录(用户登录到 Linux 操作系统中时，默认的目录) |
| HISTFILE | 命令历史文件 |
| HISTSIZE | 保存历史命令记录的条数 |
| LOGNAME | 当前的登录名 |
| HOSTNAME | 主机的名称 |

续表

| 环境变量 | 说　明 |
|---|---|
| SHELL | Shell 的全路径名 |
| TERM | 用户控制终端的类型 |
| PWD | 当前工作目录的全称 |
| PS1 | 命令基本提示符，对于 root 用户是“#”，对于普通用户是“$” |

环境变量一般都是大写的，系统启动后自动加载，可写的环境变量用户也可以随时进行修改。在脚本中，可以在环境变量名称前加上美元符号“$”来使用这些环境变量。

Linux 也提供了一些修改和查看环境变量的命令。表 3-2 列出了常用的修改和查看环境变量的命令。

**表 3-2 常用的修改和查看环境变量的命令**

| 命　令 | 说　明 | 命　令 | 说　明 |
|---|---|---|---|
| echo | 显示某个环境变量值 | set | 显示本地定义的 Shell 变量 |
| export | 设置一个新的环境变量 | unset | 清除环境变量 |
| env | 显示所有环境变量 | readonly | 设置只读环境变量 |

**【例 3-1】** 用 echo 指令显示系统提示符环境变量 PS1 的值。

```
[root@localhost ~]#echo $PS1
[\u@\h \w]\$
```

**【例 3-2】** 将某个变量设为环境变量，并查看环境变量。

```
[root@localhost ~]#export myname=geng
[root@localhost ~]#env
ORBIT_SOCKETDIR=/tmp/orbit-g
HOSTNAME=localhost.localdomain
GIO_LAUNCHED_DESKTOP_FILE_PID=3482
IMSETTINGS_INTEGRATE_DESKTOP=yes
TERM=xterm
SHELL=/bin/bash
HISTSIZE=1000
…
myname=geng
…
COLORTERM=gnome-terminal
XAUTHORITY=/var/run/gdm/auth-for-g-eF04zj/database
_=/usr/bin/env
```

新添加的环境变量

执行完 export myname=geng 语句，环境变量中就多了一个 myname，可以使用 env 指令查看所有的环境变量。